Carbon Charts: Module Guide

McMaster University
Group 18

Darsi Anandarajah
Justin Licari
Eliad Moosavi
Thomas Mullen
Richard Zhang

December 19, 2018

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**	**Revision**
17/12/2018	1.0	Created document	REV-0

1 Introduction

1.1 Purpose

This document outlines the system and component design of a data visualization library, Carbon Charts. This document will provide detailed information on the design strategies, programming principles and architectural patterns employed in the design of Carbon Charts. This document will serve as the module guide, providing module decomposition logic and the criteria used to assign responsibilities amongst major modules. Module decomposition rules are based on information hiding and described by:

- The role played by the individual modules in the overall system operation.
- The secrets associated with each module.
- The facilities provided by each module.

1.2 Scope

Our project is centered around creating a charting library for IBM's Carbon Design ecosytem of front-end modules, while ensuring it is accessible, reliable, performant, and feature-rich.

In addition to core functionality and charting types, we hope to implement:

- Bubble Charts
- Area charts (individual and stacked)
- Combo charts

In respect to accessibility, reliability, performance, we hope to implement:

- Devops and build process optimization
- Screen reader accessibility
- RTL support

- Zoom and scroll support
- Threshold support

In respect to demonstrating the library, we hope to implement:

- A micro controller with sensors feeding real time data to the library and showing it in the graphs

1.3 Development Methods

The project will be developed with the JavaScript environment NodeJS. Git will be used for version control using the GitHub client. Each group member will maintain their own up to date fork of the Carbon Charts library. Commits are merged in upon successful pull request.

1.4 Assumptions

- It is assumed that developers will use library functions correctly, and pass in the correct data types within the correct data range.

1.5 Acronyms and Definitions

1. Axis Chart: Axis charts display data between a set of axis. Axis charts are 2 or 3 dimensional, with at least an x and y axis.
2. Non-Axis Chart: Axis charts display data within an area not defined by axis. (e.g. Pie Charts).
3. Right-to-Left Support: The ability to display languages that read right to left, such as Farsi.
4. Browser Tiers: We define two tiers of browsers A and B. Tier A browsers consist of the two latest versions of Chrome, Firefox, Opera and Safari at any time, and any versions below those including IE-11+ will be part of tier B.
5. Test Data: A set of example data. This set will initially consist of a random sampling of all valid data within the expected range, but will be extended to any particular data that cause issues in the software

as it develops. Requirements that involve the test data in their fit criterion will be considered unmet if any member of the test data fail that requirement.

6. Demonstration Environment: An environment resembling a common household room; no extreme conditions (temperature, pressure, magnetic fields, impact forces etc).

7. Internalization

2 Design Strategies

Separation due to abstraction greatly reduces the complexity of software development and helps to increase the reliability of the whole system. This allows for greater flexibility in software analysis, design and development.

The charting library will respect these main design principles:

1. Object-Oriented Programming

    - The use of object abstraction to specify and reason about different components of the software will increase readability and maintainability of the software.
    - This is an extremely common paradigm, allowing developers to more easily contribute through open-source commits.

2. Encapsulation

    - Modules will restrict unnecessary external access to their internal components.
    - This reduces coupling between modules and reduces the API surface, allowing for simpler testing and reduced complexity.
    - This is implemented using file-splitting and tooling to compile bundles (see Development Process document for more information).

3. Inheritance

- All chart classes are implemented as subclasses of parent chart classes.
- This pattern maps the class hierarchy in design to a hierarchy in implementation and also minimizes duplication of code.

General Design Principles employed by the library include:

- Principle of Low Coupling and High Cohesion
- Open-Closed Principle
- Liskovs Substitution Principle
- Dependency Inversion Principle
- Interface Segregation Principle
- Law of Demeter

3 Anticipated and Unlikely Changes

3.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: Modules may need to be added as it is likely that shareholders will require new chart types for different scenarios.

AC2: Existing module functionality may be added, removed or modified.

AC3: Modules may need to be be added or fixes for Bug/Feature requests via stakeholders or the contributing community.

AC4: New sensor hardware and types of sensors may require new modules.

AC5: Combo Chart functionality will be improved.

AC6: Modules may be improved for library requirements and performance.

3.2 Unlikely changes

To avoid making modification to the module, following are unlikely to change:

UC1: Carbon Design language: IBM has not ratified their design language as any kind of standard and may decide to make modifications or clarifications to it.

UC2: Input and output devices (Input: Keyboard and mouse clicks, Output: console window)

UC3: There will always be a source of input data external to the software.

UC4: Individual chart functionality.

UC5: Library requirements (open-source, interactive, lightweight and feature rich).

4 Module Hierachy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 2. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

Level 1	**Level 2**
Behaviour-Hiding Module	M1
	M2
	M3
	M4
	M5
	M6
	M7
	M8
	M9
	M10
	M11
	M12
	M13
	M14
	M15
	M16
Software Decision Module	M17
	M18
	M19
	M20
	M21
	M22
Hardware-Hiding Module	M23

Table 2: Module Hierarchy

5 Connection Between Requirements and Design

The main motive of designing the system is to ensure that all the requirements in SRS document are met. Throughout the document the system is decomposed into modules and analyzed. The relation is listed in the Trace between Requirements and Modules table.

6 Module Decomposition

6.1 Behavior Hiding Modules

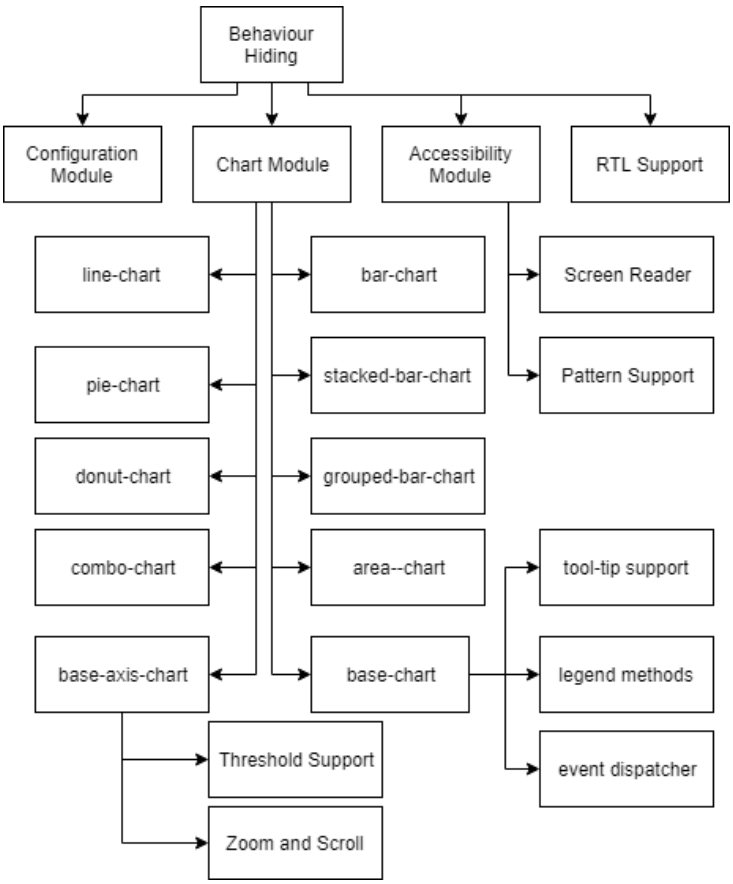

Figure 1: Behaviour Hiding Module Decomposition

6.1.1 M1: Configuration Module

Type:

Configuration Module

Secrets:

The style and configuration scheme used to provide configuration options to all charts.

Responsibilities:

Storing and maintaining options for chart configuration

Requirements:

No additional Requirements

6.1.2 M2: Base Chart Module

Type:

Chart Module

Secrets:

Canvas setup, DOM interaction, tooltip functionality, and legend functionality.

Responsibilities:

This module is responsible for providing the base functionality for all charts. Methods in Base-Chart are inherited by all charts.

Requirements:

No additional Requirements

6.1.3 M3: Base Axis Chart Module

Type:

Chart Module

Secrets:

The methods used to determine how data maps to a pixel value on a Cartesian grid.

Responsibilities:

This module is responsible for calculating the scales, setting the axes and rendering data points on axes that follow a Cartesian grid.

Requirements:

No additional requirements

6.1.3.1 M4: Threshold Support Module

Type:
Base Axis Chart Module

Secrets:
The methods and rules determining threshold support on base axis charts.

Responsibilities:
This module is responsible for determining and providing threshold support to charts derived from base axis charts.

Requirements:
FN-7

6.1.3.2 M5: Zoom and Scroll Support Module

Type:
Base Axis Chart Module

Secrets:
The methods and rules providing zoom and scroll functionality to base axis chart.

Responsibilities:
This module is responsible for determining and providing zoom and scroll support to charts derived from base axis charts.

Requirements:
FN-8

6.1.4 M6: Bar Chart Chart Module

Type:
Chart Module

Secrets:
The methods and rules determining event-handling and how data points maps to a pixel on a bar chart.

Responsibilities:

This module is responsible for getting scale information, display data, and rendering data points on axes that follow a bar chart configuration.

Requirements:

FN-12

6.1.4.1 M7: Stacked Bar Chart Chart Module

Type:

Chart Module

Secrets:

The methods and rules determining event-handling and how data points maps to a pixel on a stacked bar chart.

Responsibilities:

This module is responsible for getting scale information, display data, and rendering data points on axes that follow a stacked-bar chart configuration.

Requirements:

FN-13

6.1.4.2 M8: Grouped Bar Chart Chart Module

Type:

Chart Module

Secrets:

The methods and rules determining event-handling and how data points maps to a pixel on a grouped bar chart.

Responsibilities:

This module is responsible for getting scale information, display data, and rendering data points on axes that follow a grouped-bar chart configuration.

Requirements:

FN-12

6.1.5 M9: Line Chart Module

Type:
Chart Module

Secrets:
The methods and rules determining event handling and how data points map to pixels on line charts.

Responsibilities:
The module is responsible for determining scale and display data configurations as well as rendering data points on axes that follow a line-chart configuration.

Requirements:
FN-20

6.1.6 M10: Pie Chart Module

Type:
Chart Module

Secrets:
The methods and rules determining event-handling and how data points map to pixels on pie charts.

Responsibilities:
The module is responsible for determining scale, data representation, display data configurations and rendering data points on axes that follow a pie-chart configurations.

Requirements:
FN-14, FN-15, FN-16, FN-17, FN-18

6.1.7 M11: Donut Chart Module

Type:
Chart Module

Secrets:
The methods and rules determining event-handling, donut center, pie slices and how data points map to pixels on donut charts.

Responsibilities:

The module is responsible for determining scale, display data configurations and rendering data points on axes that follow a donut-chart configurations.

Requirements:

FN-14, FN-15, FN-16, FN-17, FN-18

6.1.8 M12: Bubble Chart Module

Type:

Chart Module

Secrets:

The methods and rules determining event-handling and how data points map to disks on bubble charts.

Responsibilities:

The module is responsible for determining scale, disk size, display data configurations and rendering data points on axes that follow a bubble-chart configuration options.

Requirements:

FN-14, FN-15, FN-19

6.1.9 M13: Area Chart Module

Type:

Chart Module

Secrets:

The methods and rules determining event-handling and how data points map to area segments between the x-ais.

Responsibilities:

The module is responsible for determining scale, area functions, display data configurations and rendering data points on axes that follow a area-chart configuration options.

Requirements:

FN-21

6.1.10 M14: Combo Chart Module

Secrets:

The methods and rules determining event-handling and how different chart types are displayed on a common axis-chart.

Responsibilities:

The module is responsible for determining the scale, tool-tip options, representation of data points by chart type, positioning of charts and rendering multiple charts on a common base-axis chart.

Requirements:

No addional requirements.

6.1.11 M15: Screen Reader Accessibility Module

Type:

Accessibility Module

Secrets:

Methods providing accessibility support to supported chart types.

Responsibilities:

This module is responsible for providing accessibility support via screen reader services for charts.

Requirements:

Usability-req-1

6.1.12 M16: RTL Support Module

Type:

RTL Support Module

Secrets:

Rules and methods determining how right-to-left support is used in chart rendering.

Responsibilities:

This module is responsible for providing RTL support for charts.

Requirements:
Compliance-req-1

6.2 Software Decision Hiding Modules

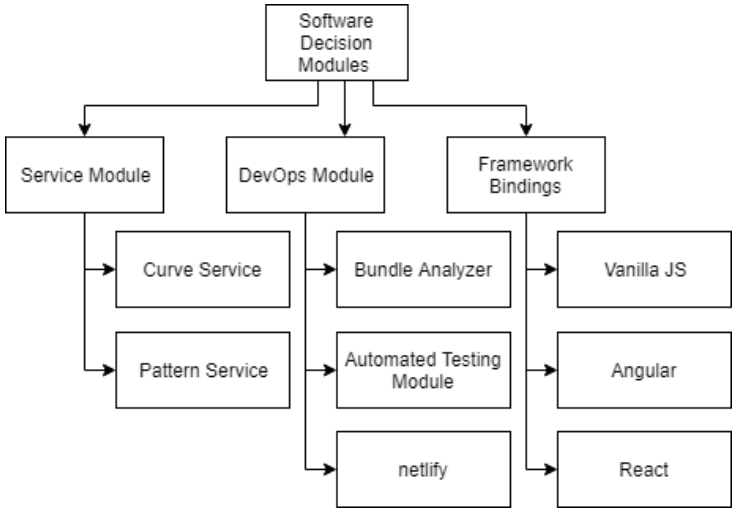

Figure 2: Software Decision Hiding Module Decomposition

6.2.1 M17: Pattern Service Module

Type: Service Module

Secrets: The methods used to provide accessibility support via patterns.

Responsibilities: Responsible for providing the pattern service to base-charts inherited by all charts.

Requirements: Usability-req-2

6.2.2 M18: Curve Service Module

Type: Service Module

Secrets: D3 curve functionality for line generators.

Responsibilities: This module is responsible for providing D3 curve service for line and combo charts.

Requirements: FN-20

6.2.3 M19: Bundle Analyzer Module

Type: DevOps Module

Secrets: The methods used to determine bundle size of the library.

Responsibilities: This module is a devOps tool responsible for calculating and reporting the bundle size of the library.

Requirements: PERF-1

6.2.4 M20: Automated Testing Module

Type:
DevOps Module

Secrets:
Rules and test cases ensuring correctness of chart functionality.

Responsibilities:
This module is responsible for providing test cases to verify and validate chart functionality.

Requirements:
No additional Requirements

6.2.5 M21: Netlify Module

Type:
DevOps Module

Secrets:
Pull request verification and validation.

Responsibilities:
This module is responsible for providing a means of verifying and validating to repository managers.

Requirements:
No additional Requirements

6.2.6 M22: Framework Bindings Module

Type:
Framework Module

Secrets:
The supported frameworks and rules determining framework bindings.

Responsibilities:
This module is responsible for framework support, bindings and framework specific configuration options. Additionally, it determines how model updating, rendering, event-handling and user interaction is bound to the framework.

Requirements:
No additional requirements.

6.3 Hardware Hiding Modules

6.3.1 M23: Communication Module

Type:

Onboard Module

Secrets:

The methods and protocol used to communicate with the hardware component.

How incoming data is serialized into JSON objects that are consumable by the charting library.

Chart configuration and event listeners that are specific to the hardware sensor array demonstration.

Responsibilities:

Abstracting implementation specifics of the hardware sensor array and it's communication channel.

Requirements:

FN-22, FN-23

7 Module Designs

7.1 Module Interface Specification

Documentation publicly available.

7.2 Base Chart Module

The Base Chart implements the core charting and rendering functionality that is inherited by all other chart types. This includes canvas setup, DOM interaction, tooltip functionality, and legend functionality. It is an abstract class and cannot be directly instantiated.

7.2.1 Interface

Uses	Configuration Module	
Constants	None	
Types	None	
Access Programs	getKeysFromData	Builds key and returns key array for legend item objects.
	getLegendType	Returns legend type for chart from configuration module
	getChartSize	Returns chart size based on chart container and clientWidth property
	updateSVG	Updates svg element
	getLegendItems	Returns legend items
	addTooltipEventListeners getFillScale getDefaultTransition getInstantTransition getFillTransition getBBox	

7.3 Base Axis Chart Module

The Base Axis Chart extends the Base Chart, adding axis functionality, zoom/scroll control, and chart scaling. It is an abstract class and cannot be directly instantiated.

Uses	None	
Constants	None	
Types	None	
Access Programs	initialDraw(data?: any)	Called when the chart needs to be drawn initially.
	update()	Populate display data, sets x and y axis and scale.
	addLabelsToDataPoints(d, index)	Adds labels to data points based on display data.
	getChartSize(container)	Returns height and width of chart size.
	resizeChart()	Repositions legend , axis-labels and scale.
	setXScale (xScale?: any)	Scales x-axis, labels and margins.
	setXAxis(noAnimation?: boolean)	Sets x-axis, labels and margins.
	setYScale (xScale?:any)	Scales y-axis, labels and margins.
	setYAxis(noAnimation?: boolean)	Scales y-axis, labels and margins.
	updateXandYGrid (noAnimation? : boolean)	Updates x and y grid upon event.

7.4 Bar Chart Module

The Base Chart extends the base-axis-chart, adding functionality for bar and stacked-bar rendering functionality and data point labeling.

7.4.1 Interface

Uses	Base-Axis-Chart Module	
Constants	None	
Types	None	
Access Programs	constructor(holder:Element, configs: any)	bar-chart constructor
	draw()	Render bar-chart
	interpolateValues(newData: any)	Creates/Update bar chart: add bars to chart, respond to events via addDataPointEventListener and dispatchEvent()
	addDataPointEventListener()	Adds toop-tip functionality specific to bar charts
	resizeChart()	Repositions legend, axis-labels and scale.

7.5 Line Chart

The Line Chart extends the Base Axis Chart, adding line/curve rendering functionality, line animation, and data point labeling.

7.5.1 Interface

Uses	Base-Axis-Chart Module	
Constants	None	
Types	None	
Access Programs	constructor(holder: Element, configs: any)	line-chart constructor
	draw()	Render line-chart
	interpolateValues(newData: any)	Creates/Update line chart: add lines to chart, respond to events via addDataPointEventListener and dispatchEvent()
	addDataPointEventListener()	Adds toop-tip functionality specific to lines charts
	resizeChart()	Repositions legend, axis-labels and scale.

7.6 Pie Chart

The Pie Chart extends the Base Chart, adding functionality to render circular charts and additional data processing to translate data into proportions.

Uses	Base-Chart Module	
Constants	None	
Types	None	
Access Programs	constructor(holder: Element, configs: any, type: string)	pie-chart constructor
	dataProcessor(dataObject: any)	Prepares data for pie chart rendering.
	draw()	Render pie-chart
	interpolateValues(newData: any)	Creates/Update pie chart: adds pie slices to chart, interpolates transitions, responds to events via addDataPointEventListener and dispatchEvent()
	addDataPointEventListener()	Adds toop-tip functionality specific to pie charts
	resizeChart()	Repositions legend, pie segments and scale.

7.7 Donut Chart

The Donut Chart extends the Pie Chart. It is exclusively a stylistic change, removing the center of the Pie Chart.

Uses	Pie-Chart Module	
Constants	None	
Types	None	
Access Programs	constructor(holder: Element, configs: any, type: string)	Donut chart constructor
	draw()	Render donut-chart
	Update(newData: any)	Update donut chart if configs are different from previous update call.
addDataPointEventListener()	Adds toop-tip functionality specific to pie charts	
	resizeChart()	Inherits resizing logic from PieChart, superclass class is encapsulated in function.

7.8 Pattern Service

The pattern service is used for the accessibility mode on all charting components. It is in charge of parsing and cleaning SVG pattern files provided to it, and injecting them into the DOM. Then it will provide an array of SVG URLs for each charting component to use.

7.8.1 Interface

Uses	None	
Constants	PATTERNS_CONTAINER:string	
Types	None	
Access Programs	constructor(holder:Element, configs: any, type: string)	Donut chart constructor
	addPatternSVGs (d:any, colorScale:any, chartContainerID:string, legendType:string)	Adds all the pattern SVGs to the container div, applying a unique ID to each one.
	getFillValues()	getFillValues()

7.9 Curve Service

This module provides D3 shape generator services. It only loads those related to supported chart types maintaining the light weight property of the library.

7.9.1 Interface

Uses	None	
Constants	curveTypes[] :d3-shape	
Types	None	
Access Programs	getD3Curve(curveName: any)	Return curve generators from d3-shape library.

7.10 Hardware Component

7.10.1 Purpose

Acquire data from sensors using an Arduino Uno and other pieces of hardware. The data will then be graphically displayed by the charts in real time. Please note every sensor uses pints 5V Vcc and ground.

7.10.2 Components

- Arduino UNO
- Breadboard
- Jumper Wires
- Resistors
- Ultrasonic Sensor
- Temperature Sensor
- Big Sound sensor

Ultrasonic Sensor
Detects the presence of a target object and measures the distance between the sensor and the object by sending a beam of ultrasound that is reflected off the object. Relevant pins:

- Trig (input): signal from Arduino to generate the ultrasound
- Echo (output): time in microseconds the sound wave travelled. Time is proportional to range of the signal. The time will be multiplied by the speed of sound to calculate distance in centimetres.

Temperature sensor
Provides temperature measurement through an electrical signal pin:

- Output pin: analog voltage reading between 0 and 1.75V.

To convert the 10-bit analog reading into voltage:
Voltage at pin in millivolts = (reading from ADC) * (5000/1024)
Then, to convert the voltage into centigrade temperature: Centigrade temperature = [(analog voltage in mV) - 500]/10

Big Sound Sensor
Sensor that detects sound intensity by converting air pressure vibrations into electrical signals. Three pins include:

- Digital output voltage: signal is proportional to sound intensity

7.10.3 Normal operation

Sensors will log data points using serial connection into a JSON file. This file will be opened and parsed to extract the data and display it graphically using the charting library.

7.10.4 Context Diagram

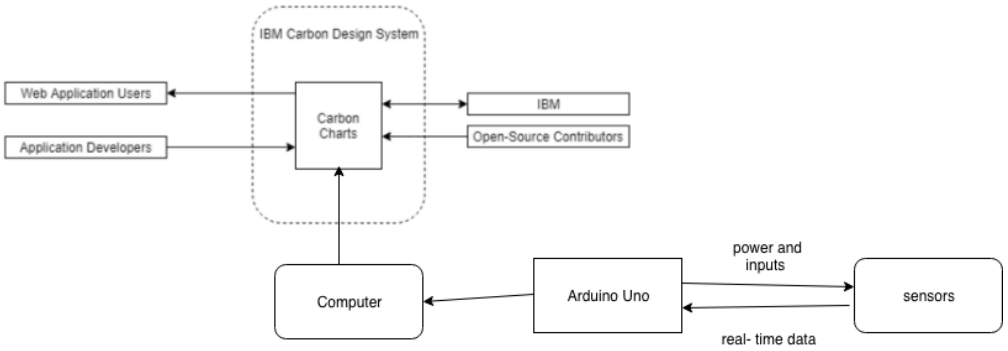

Figure 3: Context Diagram with Hardware Interfacing Component.

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	**Modules**
FN-1	M1, M2, M3
FN-2	M1, M2, M3
FN-3	M1, M2
FN-4	M2, M3
FN-5	M2
FN-6	M16
FN-7	M3
FN-8	M2
FN-9	M1
FN-10	M1, M2
FN-11	M2
FN-12	M6, M7, M8
FN-13	M7
FN-14	M10
FN-15	M11
FN-16	M10, M11
FN-17	M10, M11
FN-18	M10, M11
FN-19	M12
FN-20	M9
FN-21	M13
FN-22	M23
FN-23	M23
USB-1	M15
USB-2	M17

Table 3: Trace Between Requirements and Modules

AC	Modules
AC1	M1, M2 M3
AC2	M2, M3
AC3	M20, M21 and which ever chart/feature module is being fixed.
AC4	M23
AC5	M14

Table 4: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

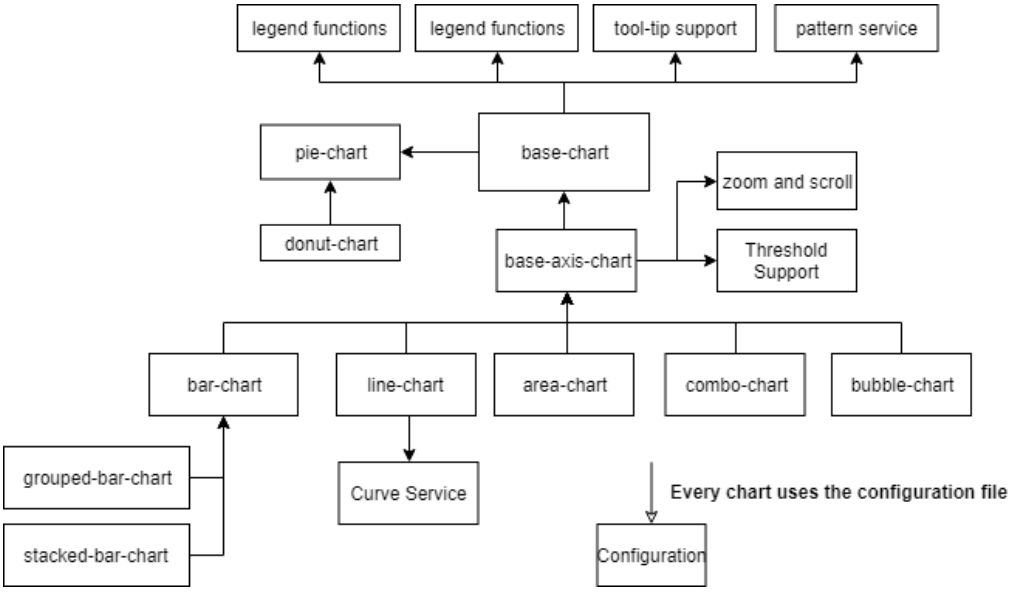

Figure 4: Hierarchy among modules

10 References

1. Parnas, D.L. "Designing Software for Ease of Extension and Contraction", Proceedings of the Third International Conference on Software Engineering, pp. 264-277, 10-12 May, l978.

2. Parnas, D.L., Paul, C.C., David, M.W. "The Modular Structure of Complex Systems", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING,VOL. SE-1 1, NO. 3, MARCH 1985

3. Parnas, D.L., Britton, K.H. "A Procedure for designing abstract interfaces for device interface modules.", ICSE '81 Proceedings of the 5th international conference on Software engineering, pp. 195-204, 09 - 12 March, 1981.